## 2. Cellular automata

### *2.1. The game of life*

The evolution of the Game of Life on a lattice is fully deterministic and is set by the following set of rules:

- Any live cell with less than 2 live neighbours dies.
- Any live cell with 2 or 3 live neighbours lives on to the next step.
- Any live cell with more than 3 live neighbours dies.
- Any dead cell with exactly 3 live neighbours becomes alive.

**Note that in Game of Life, it is conventional to consider as neighbours all cells which have at least a point in contact with a given cell; i.e. there are 8 neighbours for each cell (north, south, east, west, and the four neighbours along the lattice diagonals). This is different from the Ising model, and the SIRS model considered below, where we only consider the four nearest neighbours of a cell.**

1. Write a Python code to simulate the game of life on a $50 \times 50$ 2d square lattice with periodic boundary conditions. Your program should allow the user to choose between a random initial condition and one in a set of selected initial conditions, which should include a state developing into an oscillator and a state developing into a glider (or another perpetually moving structure). You should consult the lecture notes when building such selected initial conditions. Your program should also show a visualisation of the state of the system as it is running.

2. For the random initial condition, you can monitor the system evolution by measuring the number of active sites as a function of time. At a given point, the number typically stops evolving: the system has reached either an equilibrium, which normally involves oscillators (such as the "blinker" described in the lecture notes) and absorbing states (such as the "beehive" described in the lecture notes). By measuring the number of active sites over time and the point at which this stops evolving, you should plot a distribution of the times needed to equilibrate (you will probably need of the order $\sim 100$ simulations to get a good histogram/distribution, but each simulation is pretty quick!).

3. For the glider, code up a function which tracks the position of its centre of mass as a function of time. The centre of mass of a set of lattice points $\{\mathbf{r}_i\}_{i=1,\ldots,n}$ is given simply by $\mathbf{r}_{\rm cm} = \frac{\sum_{i=1}^{n} \mathbf{r}_i}{n}$. However, you should account for periodic boundary conditions, for example by discarding data points for which the glider is crossing some of the boundaries. Use the centre of mass data to estimate the speed of the glider. [If you like, you

can replace for this exercise the glider with any other moving structure which you may have found.]

*2.2. The SIRS model for epidemics spreading*

The SIRS model comprises agents (i.e., people or other organisms) arranged on a two-dimensional lattice. The agents may contract some infection (e.g., the annual flu), and can be in one of three states:

- $S$: susceptible to the infection;
- $I$: infected;
- $R$: recovered from the infection (and hence immune to it).

The SIRS model is called like this because agents go through these states cyclically: $S$ changes to $I$ when a susceptible agent comes into contact with an infected one; recovery changes the agent from $I$ to $R$; eventually, in the absence of contact with infected agents, the immune response wears off and $R$ changes back to $S$.

More precisely, the approach which we shall use in this Checkpoint to simulate this model is through a random sequential updating scheme. Here, as in the Ising model, only one site is updated in each timestep, and the site to update is chosen randomly. Again as in the Ising model, if there are $N$ sites in the system, $N$ such updates is called a sweep. The set of rules which define the (stochastic) SIRS model is as follows,

- $S \to I$ with probability $p_1$ if at least one neighbour of the site to update is $I$; otherwise the site is unchanged.
- $I \to R$ with probability $p_2$.
- $R \to S$ with probability $p_3$.

1. Write a Python code to simulate the SIRS model on a $50 \times 50$ 2d square lattice with periodic boundary conditions, (so that $N = 50^2 = 2500$). Your program should allow the user to choose the system size, probabilities $p_1$, $p_2$ and $p_3$, and use the random sequential updating scheme. As usual, it should also be able to show a visualisation of the state of the lattice as it is running.

2. Find some parameter sets which leads to (i) an absorbing state with all sites susceptible to the infection; (ii) a dynamic equilibrium between $S$, $I$ and $R$; (iii) a case with cyclic wave of infections through the lattice. Visualise the dynamics in each case.

3. Now set $p_2 = 0.5$. The average number of infected sites, $\langle I \rangle$, is a quantity that provides a quantitative measure of the different possible behaviours (just as the magnetisation in the Ising model). Using this measure, obtain

the phase diagram of the system in the $p_1 - p_3$ plane (i.e., a diagram showing the regions where qualitatively different behaviour emerges). To do this, you should plot, in the same $p_1 - p_3$ plane, the value of the average fraction of infected sites $\langle I \rangle / N$, e.g. as a contour or colour plot. To do the averaging, for each value of $(p_1, p_3)$, you can use a sufficiently long simulation after equilibration: you can use 100 sweeps as equilibration time, and a runtime of 1000 sweeps (with measurements every sweep) will be sufficient for this problem. Please use a resolution of 0.05 or finer in both $p_1$ and $p_3$.

4. Where are the waves in your phase diagram? To pinpoint these, one way is to show a countour plot of the variance of the number of infected sites $\frac{\langle I^2 \rangle - \langle I \rangle^2}{N}$ as a function of $p_1$ and $p_3$ (still for $p_2 = 0.5$). As you will see, this graph tends to be very noisy: we would need to average it over many runs to smooth out the statistical fluctuations, which is impractical given our time constrains. For this reason, you should also compute this variance along a cut at fixed $p_3 = 0.5$, between $p_1 = 0.2$ and $p_1 = 0.5$ – the reduced parameter space allows longer runs (10000 sweeps) which you can use to get a more accurate plot. You should add error bars to this last plot. You can do it quite simply with the code you've already got! For instance you can use a resampling method.

5. Modify your program so that some fraction of agents are permanently immune to the infection (i.e., in the $R$ state), modelling for instance infinitely efficient vaccination. For $p_1 = p_2 = p_3 = 0.5$, what fraction of immunity – $f_{\mathrm{Im}}$ – is required to prevent the spreading of the infection? A good way to address this question is via a plot of the average infected fraction versus $f_{\mathrm{Im}}$, and here as well you should add error bars. Here the recommended way to get error bars is by averaging over multiple (say 5) runs and taking the standard error of the mean.

*2.3 Figure and datafile checklist for checkpoint*

For convenience, the basic plots and datafiles which you should have for the quantitative analysis for this checkpoint are given below (these correspond to the graphs and datafiles mentioned in the marksheet).

1. For the Game of Life, you should have a histogram of the time needed to reach an equilibrium for simulations starting with a random initial condition, and either a plot of the centre of mass versus time for the glider initial condition together with appropriate fits, or another calculation of the glider speed.

2. For the SIRS model without immunity, you should have a contour (heatmap) plot of the average infected fraction and of the scaled variance of infected sites versus $p_1$ and $p_3$, for $p_2 = 0.5$ (as resolution, you should change $p_1$ and $p_3$ in steps of 0.05 or lower). You should also have a plot of the variance of infected sites versus $p_1$ for $p_2 = p_3 = 0.5$, with error bars.

3. For the SIRS model with immunity, you should have a curve showing the average infected fraction of sites as a function of the immune fraction $f_{\mathrm{Im}}$, showing error bars, for the specified parameter set.

4. For each plot, you should have a corresponding datafile.